Software Engineering Project “Bugfind”
Maintenance Document

Project group
Perttu Haimi, Petteri Hintsanen, Philipp Masche, Toni Ruokolainen and Osku Salerma

Instructor
Peter Lamberg

Helsinki 9th May 2001
Maintenance Document
UNIVERSITY OF HELSINKI
Department of Computer Science

Version history

- v0.1, 25.3.2001 (first draft)
- v0.9, 4.5.2001 (first proposed version)
- v1.0, 8.5.2001 (final version)

Contents

1 Preface

This paper is aimed to give some background information about the Bugfind system, so that the maintenance, further improvements, etc. can be done more easily. Basically it discusses about issues from design, implementation, testing and distribution. It also contains a To-Do list of ideas how to improve the Bugfind system.

2 Design

The design was dominated by the problem of distribution. Fortunately all other tasks, which are mainly file and image processing are not complicated, so there is not much to say about the design concept in the whole. It is straightforward and simple.

3 Implementation

During the implementation we did no weird tricks but programmed the whole program quite conventionally, sticking into the modular design specified in the design document. Since this project is an exercise in the software engineering, things are not always 100% perfect. For example, some functions are clearly unoptimized and the algorithms used may be not the best ones available. See the To-Do list for further discussion.

3.1 Code modules

The source code is kept quite functional and modular; we placed the source of each module of the design into a separate source file with it's own header file. All source files are placed in the subdirectory `./src/bugfind/src`, all header files in `./src/bugfind/include`, where the `./` represents the source directory of the Bugfind project CVS tree.

3.2 Compiling

To compile Bugfind, run `make` in the `./src/bugfind` directory. Again the `./` represents the source directory of the Bugfind project CVS tree.

3.3 Optimization

Our task was to engineer a correctly working program, optimization was not recommended in the first place. All algorithms we use are working correctly, but not necessarily optimal concerning the runtime and the memory usage.

The problem of finding particles in images is solved in Bugfind with brute force approach. There are for sure a lot of small optimizations possible in this approach and totally different ways may solve this problem more efficiently.

3.4 Reusability

Even if it was not expected from us to implement high reusable code, our modular programming style makes reuse of single modules or functions of Bugfind easy. It is also easy to replace single function by another, e.g. for the sake of optimization.

4 Testing

To test a program like Bugfind is on the level of algorithmic correctness easier than it to test and predict the behaviour of bugfind in practise. We tried to do thorough module testing, and to make some validation tests that we thought could be relevant, for instance the effect of noise on simulated EM-images to correlation scores that Bugfind produces.

4.1 Module testing

The module testing is described in detail in the testing document. When we defined the test cases, we tried to cover all failure possibilities and removed all known mistakes. Nonetheless, almost surely there exists some unknown bugs. A completely bug-free program of this scale is difficult to produce.

4.2 Validation testing

We could test Bugfind with "artificial" test data, which we created with tools that we developed, and with the "real" data we got from our customer. Of course, the true usefulness of Bugfind will only be found out when it is actually put in use. The performance depends a lot on a good choice of the parameters and the quality of the input images.

5 Distribution

The distribution scheme is described thoroughly in the design document. After preliminary tests, we learned, that the idea of dividing the tasks between the slave nodes by using a group of projection angles as the unit of work, produces a nice trade-off between a high distribution level and a heavy network traffic.

6 To-Do list

This section contains some points where a better design and/or implementation would probably make Bugfind more reliable, efficient or simply just more convenient for the user. Most of the ideas have emerged during the testing phase, and thus were practically impossible to be implemented then due to the tight project schedule.

6.1 Interpolation module

A better interpolation algorithm could produce better scaling results and thus improve the correlation value in the comparison phase, resulting in more accurate matches, specifically with very large macromolecules. The current linear interpolation method is very simple and the quality is not the best possible.

6.2 Projecting and rotating

Sometimes clipping occurs when rotations or projections are performed. For example, if some image is rotated by 45°, there will be substantial amount of clipping if the image doesn't contain enough blank space on the sides to "hold the corners" after rotation. It would be convenient to simply assume that the rotation/projection module creates required padding by enlarging the image before rotation or projection.

6.3 Scaling

As now implemented, the scaling of the projections of the model is calculated from user parameter `diameter` (which affects the radius of the gaussian filtering and maximum finding step) and from the lenght of the longest edge of the model box. Note: in this discussion the model box means "the box around the model" and the model means the pixels in the box that actually corresponds to the particle that the model represents.

This approach assumes that the `diameter` setting and the longest edge of the model box corresponds to the same diameter of the model. Let us assume that the model is exactly of the correct size, but has for some reason empty voxels around it so that the edge of the model box is twice the diameter setting. In this case, Bugfind will scale the projections to half of their original size and the matching will be poor.

Even if the model snugly fits the model box, the `diameter` setting and the longest edge of the model box may not agree. Let us assume that the user builds the model so that the longest diameter of the model is aligned to the space diagonal of the model box. The worst case situation is then a rod-shaped model and a cubic model box. In any case, the longest edge of the model box will be greater than or equal to the longest diameter of the model divided by $\sqrt{3}$. Let us again assume that the longest diameter of the model and the `diameter` setting agree, so no scaling is needed. Nevertheless, Bugfind will in worst case scale the model to $\sqrt{3} \approx 1.73$ times the original size.

The approach is also inflexible if the user wants to use different `diameter` setting than the largest diameter of the particle. For instance, if the user wants to try whether using the smallest diameter of the particle as the `diameter` setting, improves the results.

One solution would be to introduce another parameter, `scaling_factor`, which would leave the responsibility of the correct scaling to the user. Another possibility would be to implement an algorithm which finds the largest diameter of the model. This would still leave the problem whether the user really ment that the `diameter` setting corresponds to the largest diameter of the particle (and hence model).

6.4 Pruning "empty" squares

Currently, the gaussian filtered EM-image is divided to squares with edge length being the same as the `diameter` setting. The maximum value of each square is then sought and added to a list. It is possible that fairly large number of squares do not contain real particles, and it would speed up things if these could be ignored. For this reason, some kind of predictive measure of usefulness of a square is needed, and for this relative variance might be suitable (relative here means that the variance of a square is divided by its mean). When the variance of the square is calculated from the filtered image, it should not be very sensitive to noise. Also, it is a local measure. The variance can be calculated in one pass over the square by the equation $D^2(X) = E(X^2) - (E(X))^2$. This would intruduce one more user parameter, `var_limit`. If the relative variance of a square would be less than `var_limit`, its maximum would not be passed to the slave nodes for comparison.

6.5 Skipping zero-intensity pixels of projection

In practice all biological objects (which indeed very rarely have the shape of rectangular parallelepiped), the model box will have voxels which do not correspond to the object (these may or may not be set to zero). Also, to avoid clipping in the projection stage, we introduce some empty space around the original model box. In current implementation every pixel of the projection of the model will be used in the calculation of the correlation coeffiecient between the projection of the model and the area of interest on the electron micrograph image.

This may lead to problems since the surrondings of the particle (nearby particles, specs of dust, etc.) will affect the correlation coefficient. One way to deal with this would be to add an `if` statement to inner loop of comparison module, which would leave out every pixel where the value of projection image is zero (or "empty"). Another way, which might speed up the comparison, would be to preprocess the projections so that each line has a pointer to the first nonzero pixel on this line, and an integer value which tells how many pixels to compare on this line. This preprocessing could be done after the rotations have been precalculated. Since this would only be done once, the number of comparisons should decrease and the method should give some speedup when the number of candidate points is large enough.

6.6 Optimization approach

Bugfind does an exhaustive search through all projection angles (with given density) of the model for each maximum of the micrograph. This takes a fairly large amount of calculations and processing time, one candidate point is in the order of 5 to 30 seconds, depending on the size of the model. It is likely that the candidate points produced by gaussian filtering are not true centerpoints of the particles. For instance, neighbouring particles may pull the maximum given by gaussian filtering away from the true center of the particle. Another problem may occur if the model is asymmetric in the sense that the center of mass is not in the center of the model. Since gaussian filtering will have the maximum values at the center of mass, Bugfind will be comparing apples to oranges.

To counter these problems, Bugfind searches around the candidate point, doing exhaustive search for each point that has distance less than or equal to the parameter `searchradius`. This means that the number of points to consider is approximately πs^2, where s is the search radius. In one test we had an image with size 2000×2000 pixels. The image had spherical particles of diameter 300 very close to each other, with particle distances around 50 pixels. For this data, gaussian filtering gave very bad candidate points, which could miss the center of the particle by 100 pixels. When considering that the calculation at these candidate points took approximately 10 minutes, increasing the search radius to find the true centerpoints would increase the time taken to something like 200 days.

Better approach would be to use function optimization methods to find the true centerpoints with a smaller number of calculations. This is essentially a discrete two-dimensional optimization problem. If one allows interpolation in comparisons, it becomes non-discrete. Maybe one could find useful methods from the literature [Bre73]. The success of this approach depends on the roughness of the optimisation space, ie. how easy it is to get stuck into a "wrong" local optimum.

The orientation angles can also be considered as variables, which leads to a five dimensional (three angles and the x and y coordinates) optimization problem. The advantage would be (if the approach works, that is) further decrease in the number of comparisons needed. The distribution scheme would have to be changed from dividing the angles to dividing the candidate points, since then all nodes would need access to all angles. Also, in current implementation, each projection is calculated exactly once for all candidate points and compared several times, whereas optimization methods would make similar reuse of projections difficult.

6.7 Replacing the correlation comparison

Our solution for the comparison of the projections with the micrograph image is based on the (Pearson) correlation value between the compared elements. However, there do exist other matching techniques which could be used as a replacement for the correlation method. Clearly, due to the noise involved, some kind of approximate matching algorithm would be needed. Deciding the "similarity unit" is another question. Again many options are available, from the standard editing distance to the Hamming distance, for example.

6.8 MPI multicasting

MPI-multicasting could be used for the data transfer between the master node and the slave nodes during the preprocessing. The current MPI-implementation used [MPIb] does not implement an adequate multicasting function, so we did not have a chance to use multicasting for now. However, future versions of the MPI implementation might have this feature, and very probably the commercial versions of the MPI standard [MPIa] do have it already.

6.9 More utility programs

Small utility programs would be useful for viewing the program performance and results. For example, a small program for extracting the found particles from the image into separate files would be nice to have around while testing and improving the software.

7 Utilities

Few small utility programs have already been produced during the implementation phase. These tools can be used for e.g. module testing and visualizing. Possible further developers will probably find them interesting, but due to the lack of testing and documentation, they are not recommended for the end users of Bugfind. To compile the utilities, run `make` at the `./src/tools` directory, where `./` points to the root directory of the Bugfind CVS.

7.1 mrc2pgm / pgm2mrc

These programs are simple image format converters. `mrc2pgm` converts an image file from the *mrc*-format to the *pgm*-format, and `pgm2mrc` does the same vice-versa.

```
SYNOPSIS
        mrc2pgm in.mrc out.pgm
        pgm2mrc in.pgm out.mrc
```

7.2 mrcscale

This program demonstrates the scaling function. It scales the given image to the given size, using linear interpolation for anti-aliasing. Use in conjunction with `mrc2pgm` and `xv` to visualize the results.

```
SYNOPSIS
        mrcscale in.mrc out.mrc new_width new_height
```

7.3 mrcproject

This program demonstrates the projecting function. It projects the given model from the desired angle and outputs the projected 2D image. Use in conjunction with `mrc2pgm` and `xv` to visualize the results.

```
SYNOPSIS
        mrcproject alpha beta in.mrc out.mrc
```

7.4 rod

This program creates a small rod-like artificial 3D model. The model can then be used for e.g. generating test pictures. The program outputs one file, `rodmodel.mrc`, containing the model data.

```
SYNOPSIS
        rod
```

7.5 test

This (unintuitively named) program generates multiple projections from the given model and saves the projections to a *png*-file for further inspection. The density of the projections is given as a parameter.

```
SYNOPSIS
        test density model.mrc output.pgm
```

7.6 testgen

This program generates simple test images.

```
SYNOPSIS
        testgen [options] model1.mrc ...

DESCRIPTION
        Options are:

        -wN     width (N pixels) of the resulting picture
        -hN     height (N pixels) of the resulting picture
        -nN     Number of particles
        -sN     random seed
```

```
-fN      filter the model with limit N
-tR      background level at top of the picture ]0-1[
-bR      background level at bottom of the picture ]0-1[
-rR      signal to noise -ratio R <= 0 => no noise
-c       noise with constant distribution
-g       noise with gaussian distribution
-i       invert the image
-m       normalize the model
-oFILE   output filename
```

7.7 visres

This program marks the output from Bugfind's output file into the given *mrc* image. Matches from the output file are marked as small crosses. The cross' intensity represents the quality of the match; the better the match, the more brighter the cross.

```
SYNOPSIS
        visres [options] output_file image.mrc

DESCRIPTION
        Options are:

        -lR     limit (R) for correlation coefficient
```

References

[Bre73] R.P Brent. *Algorithms for Minimization without Derivatives.* Englewood Cliffs, NJ: Prentice Hall, 1973.

[MPIa] MPI: Message Passing Interface. WWW hypertext, http://www.mpi-forum.org.

[MPIb] MPICH: A portable implementation of MPI. WWW hypertext, http://www-unix.mcs.anl.gov/mpi/mpich.